TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Disseny i avaluació via simulació d'un sistema de localització de dispositius en un entorn de topologia canviant.

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica.

AUTOR: Víctor Romero Guardiola.

DIRECTOR: Anna Agustí Torra.

DATA: 15 de febrer de 2013.

Title: Disseny i avaluació via simulació d'un sistema de localització de dispositius en un entorn de topologia canviant.

Author: Víctor Romero Guardiola.

Director: Anna Agustí Torra.

Date: February, 15th 2013.

Overview

During the last years, the number of mobile devices has increased exponentially. This fact has triggered the development of new applications and, with them, the need for designing new communication systems and protocols to make them viable.

This work proposes the design of a system intended to locate the position of the animals during the mountain pasture season. The aim is to provide a simple tool to facilitate the work of the pastors that develop their job in rural environments of difficult access and with low connectivity using low energy-consuming technologies and protocols specifically designed for sensors networks, such as the Routing Protocol for Low Power and Lossy Networks (RPL).

Títol: Disseny i avaluació via simulació d'un sistema de localització de dispositius en un entorn de topologia canviant.

Autor: Víctor Romero Guardiola.

Director: Anna Agustí Torra.

Data: 15 de febrer de 2013

Resum

En els darrers anys s'ha incrementat exponencialment el nombre de dispositius mòbils des d'on rebre i enviar dades. Aquest fet ha desencadenat el desenvolupament de noves aplicacions i, amb elles, la necessitat de dissenyar nous sistemes i protocols de comunicacions per a fer-les viables.

En aquest treball final de carrera es desitja dissenyar i avaluar via simulació un sistema que permeti representar de forma aproximada la ubicació dels animals de pastura d'alta muntanya. L'objectiu és proporcionar una eina que simplifiqui les tasques dels ramaders que desenvolupen la seva feina en entorns rurals de difícil accés i amb baixa connectivitat utilitzant tecnologies de baix consum energètic i fent ús dels protocols dissenyats per a xarxes de sensors, en particular, el protocol *Routing Protocol for Low Power and Lossy Networks* (RPL).

“Una de les coses que sempre m’han recordat a casa és de ser agraït; per tant, en aquest TFC no podia faltar una petita menció a tota la gent que m’ha donat suport i que ha fet possible que aquest TFC estigui ara acabat:
Primer de tot, vull agrair a la meva tutora Anna Agustí Torra la seva paciència constant i el seu suport al llarg de tot el projecte, ja que sense ella no hauria estat possible acabar-lo.
Vull donar les gràcies també a tota la meva família, per aguantar-me i donar-me suport en els bons i mals moments que han estat presents en la realització del TFC i al llarg de tota la carrera en general, i per l’esforç que ha comportat per a ells que pogués realitzar-la.
També vull agrair a tots els meus companys i amics de la EPSC tots els anys i moments que hem passat plegats, i als meus amics de sempre per donar-me suport i moments necessaris de tranquil·litat. I en especial al meu company i amic Víctor De Blas Martín, la seva ajuda en alguns punts on no sabia com continuar i a la meva amiga Irene Alboquers Hervàs per la segona revisió del text escrit.
I sobretot, vull donar les gràcies a la meva parella, Laura Pellín, per aguantar-me en els pitjors moments que s’han hagut de passar en aquest TFC i per recordar-me que sense esforç i sacrifici no es poden realitzar les coses més complicades.
A tots vosaltres, moltíssimes gràcies per tot.”

ÍNDEX

ÍNDEX DE FIGURES

CAPÍTOL 1. INTRODUCCIÓ

Durant els últims anys s'ha incrementat exponencialment el nombre de dispositius mòbils que poden rebre i enviar dades des de diferents posicions. Aquest fet ha propiciat l'aparició de xarxes amb topologies dinàmiques i canviants i, amb elles, la necessitat de dissenyar i desenvolupar sistemes de comunicació que permetin proporcionar diferents serveis als usuaris d'aquests dispositius.

Aquest treball final de carrera proposa un sistema de localització del ramat de pastura utilitzant una tecnologia de baix cost energètic i fent ús del protocols dissenyats per a xarxes de sensors (en particular, el protocol *Routing Protocol for Low Power and Lossy Networks*, RPL) per tal de proporcionar un mecanisme simple i útil per als ramaders que desenvolupen la seva feina en entorns rurals de difícil accés i amb baixa connectivitat.

Per tal de definir el sistema s'han hagut de completar les fases següents:

- *Estudi de l'interès de l'usuari i requeriments de l'aplicació.* Es va realitzar un estudi sobre els usuaris finals de l'aplicació i les seves necessitats, així com de la localització on es pretenia desplegar el sistema i els requeriments i limitacions que les particularitats de l'entorn imposaven a l'hora de plantejar-ne el disseny.

- *Estudi de les tecnologies.* Per al disseny calia determinar quines tecnologies i protocols existents s'havien de tenir en compte, així com les eines de simulació que es podien emprar per tal de provar i validar el sistema proposat.

- *Disseny de l'aplicació i avaluació via simulació.* Un cop determinats els requeriments del sistema i dels usuaris i, escollides les tecnologies i eines que s'utilitzarien per a validar el sistema proposat, es va procedir a fer-ne el disseny, a programar el codi necessari i a simular diferents escenaris que permetessin verificar-ne el funcionament.

- *Conclusions i línies futures.* A partir dels resultats obtinguts a través de la simulació, s'han establert possibles línies futures per tal de definir un sistema que es pugui provar en un entorn real.

Aquesta memòria està estructurada de la següent manera. Al capítol 2 es descriu el sistema a dissenyar. Al capítol 3 s'introdueixen les nocions bàsiques de xarxes de sensors sense fil i es descriu el funcionament del protocol RPL. Al capítol 4 s'explica el simulador Cooja, utilitzat per a implementar el sistema que es descriu al capítol 5 i per a fer les simulacions que es descriuen al capítol 6. Al capítol 7 s'enumeren les conclusions i les futures línies de treball.

CAPÍTOL 2. OBJECTIUS I ABAST

En aquest treball final de carrera es vol dissenyar un sistema que permeti determinar de forma aproximada la ubicació dels animals a la muntanya durant els mesos de pastura.

La idea sorgeix arran de parlar amb diversos ramaders de les muntanyes del Ripollès, una zona muntanyosa i de difícil accés. Els ramaders ens han explicat que durant un període de 3 o 4 mesos, els animals pasturen per la muntanya, en una zona d'entre 1 i 5 Km de diàmetre, sense que ells tinguin un control exacte de la seva posició. Aquest desconeixement fa que, per exemple, els propis ramaders no tinguin constància de si algun animal necessita assistència veterinària o de si hi ha algun depredador que pugui estar amenaçant la seguretat del bestiar (cosa que podrien deduir a partir del patró de moviment dels animals).

El sistema que es proposa consisteix en col·locar sensors als animals de manera que es comuniquin entre sí fins a fer arribar la informació a un dels punts fixos situats de manera estratègica a l'entorn. Així, unint la informació recollida pels diferents punts es podria crear un mapa de la posició aproximada dels animals.

El perímetre de la zona de pastura acostuma a estar delimitat per tanques (en alguns casos electrificades) o per la pròpia orografia del terreny (barrancs, penya-segats o rius). En el cas de tractar-se de tanques on ja hi ha arriba corrent elèctric, es podrien situar nodes fixos alimentats per la pròpia corrent de la tanca. D'altra banda, cada animal hauria de portar un dispositiu (per exemple a l'esquellot) de baix consum energètic amb una bateria que es carregués amb energia solar. El radi de transmissió dels dispositius que portarien els animals no permetria comunicar-se directament amb els nodes fixos de les tanques i per això caldria pensar en un sistema de comunicacions multi-salt que permetés l'intercanvi d'informació entre els nodes mòbils (els animals), fins a fer-la arribar als nodes fixos i finalment al punt central d'anàlisi i representació de les dades. La **Fig. 2.1** mostra un esquema del sistema proposat.

Per a comunicar els animals entre sí i fer arribar la informació fins al punt d'accés (node fix) més proper, es proposa utilitzar el protocol RPL (*Routing for Low-Power and Lossy Networks* **[1]**), que es basa en construir arbres d'encaminament arrelats en un punt (que seria el punt d'accés).

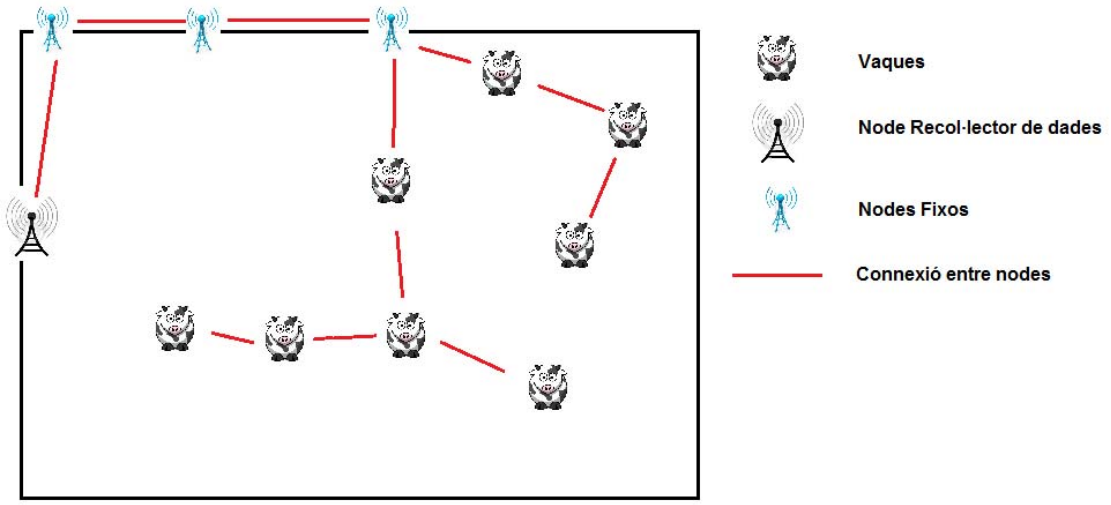


Fig. 2.1: Esquema del sistema proposat.

És possible que per a poder cobrir tota la zona no n'hi hagi prou amb els nodes fixos situats al tancat del perímetre i, en tal cas, caldrà situar dispositius fixos addicionals en posicions estratègiques, tal i com s'il·lustra a la **Fig. 2.2**.

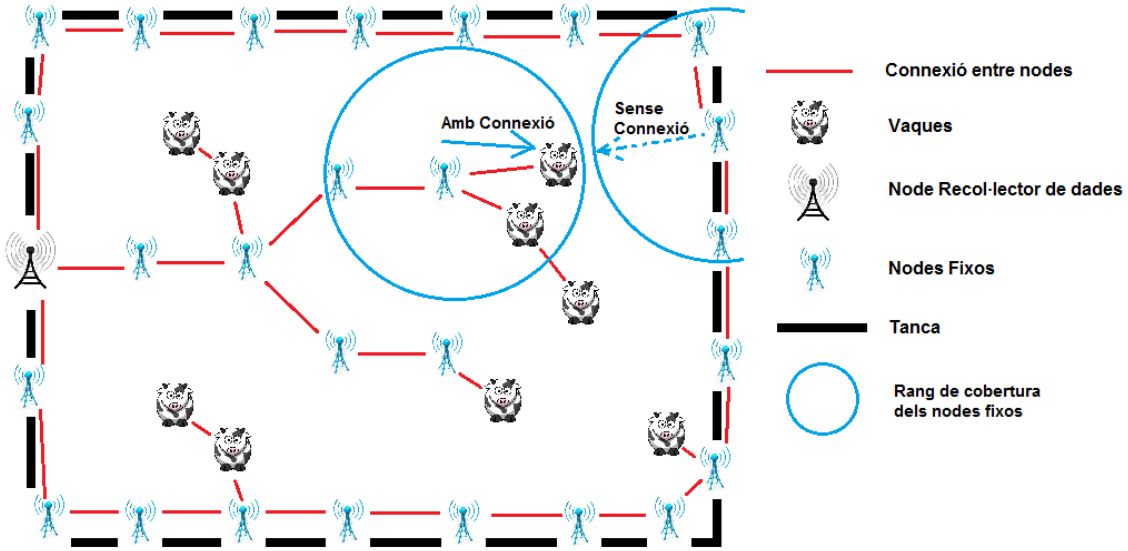


Fig. 2.2: Esquema amb nodes fixos addicionals.

CAPÍTOL 3. EL PROTOCOL RPL

Les xarxes de sensors sense fils (*Wireless Sensor Networks*, WSN) estan formades per sensors autònoms intel·ligents, en general alimentats per una bateria, que mesuren certes característiques de l'entorn, tals com la temperatura, la pressió, el nivell de soroll, etc. Aquests sensors (sovint anomenats "*motes*") es comuniquen via ràdio amb els seus veïns per enviar les seves mesures a un punt central de recol·lecció de dades des d'on les dades s'envien a un centre d'anàlisi i processat. L'ús de bateries i comunicacions ràdio permet minimitzar el cost de desplegament d'aquests sistemes de mesura distribuïts. El rang d'aplicació de les WSN és molt ampli. Per exemple, monitorització del clima, detecció d'incendis al bosc, monitorització de la fauna salvatge, serveis d'avís per a persones grans que viuen soles, etc **[2]**.

Les primeres xarxes de sensors estaven aïllades o es connectaven a Internet a través d'un dispositiu que actuava a mode de passarel·la (*gateway*). Utilitzaven protocols de comunicacions molt específics i sovint poc estructurats que dificultaven l'ús de dispositius de diferents fabricants.

Gràcies a l'estandardització de l'IEEE 802.15.4 (que defineix el nivell físic i el control d'accés al medi en xarxes sense fils d'àrea personal amb baixes taxes de transmissió) i a l'adveniment de l'IPv6 (que permet un rang d'adreçament molt superior al que proporciona IPv4) es van establir les bases per a una estandardització de les WSN que en facilités la seva integració a Internet.

Amb aquest objectiu, l'any 2005, l'*Internet Engineering Task Force* (IETF) va definir el grup de treball 6LoWPAN (*IPv6 over Low Power Wireless Personal Area Networks*) per estandarditzar l'ús de l'IPv6 sobre els sistemes IEEE 802.15.4. L'encaminament estava fora de l'abast del 6LoWPAN i, per aquest motiu, l'any 2008 es va formar un altre grup de treball, anomenat ROLL (*Routing over Low-power and Lossy networks*) per tal d'estudiar els protocols d'encaminament que s'utilitzaven a Internet i aquells específicament dissenyats per a WSNs. Després de concloure que cap dels protocols existents cobria tots els requeriments de les WSN, el ROLL va proposar un nou protocol anomenat RPL (*Routing for Low Power and Lossy Networks* **[1]**).

3.1. Funcionament del RPL

El protocol RPL es basa en la construcció d'un arbre d'encaminament que defineix un graf dirigit acíclic orientat a destinació (*Destination-Oriented Directed Acyclic Graph*, DODAG). Un graf dirigit acíclic (*Directed Acyclic Graph*, DAG) té la propietat que tots els arcs estan orientats de manera que no hi ha cicles. Un DODAG és un DAG on tots els arcs formen part de camins orientats cap a l'arrel de l'arbre (node passarel·la o *gateway*). La construcció del DODAG s'inicia a l'arrel mitjançant l'enviament de missatges DIO (DODAG *Information Objects*) a tots els nodes accessibles directament des de l'arrel. Un DIO conté

l'identificador i els paràmetres de configuració del DODAG i el *rank* del node que envia el DIO. El *rank* indica la posició relativa d'un node respecte a l'arrel (els nodes més propers tenen un *rank* menor). Inicialment tots els nodes tenen *rank* infinit, excepte a l'arrel, que s'inicialitza amb *rank* 0 (mirar referencia càlcul del *rank* **[3])**. Quan un node rep un DIO, compara el *rank* que conté el DIO amb el seu propi *rank*. Si el *rank* del DIO és superior, el DIO es descarta. En cas contrari, se selecciona el node emissor del DIO com a "pare preferent", s'actualitza el *rank* i, després d'un determinat retard, s'envien missatges DIO a tots els veïns als que es pot accedir directament per tal de difondre la informació. Aquest procés continua fins que tots els nodes estan inclosos al DODAG.

Utilitzant el "pare preferent" com a porta per defecte (*gateway)*, tots els nodes de l'arbre es poden comunicar amb l'arrel. Per establir les rutes en direcció contrària s'utilitzen els missatges DAO (*Destination Advertisement Object*). Es defineixen dos modes de funcionament: *storing* i *non-storing*. En mode *storing* tots els nodes del DODAG mantenen una taula d'encaminament, de manera que cada node envia missatges DAO informant de les adreces IPv6 que coneix al seu "pare preferent". En mode *non-storing* només l'arrel de l'arbre manté la taula d'encaminament i, per tant, els missatges DAO s'envien en mode *unicast* a l'arrel.

Una vegada format l'arbre, hi ha diversos factors que poden modificar la composició del DODAG, tal i com s'explica en els següents subapartats.

3.1.1. Introducció de nous nodes

Per tal que un node es pugui afegir a un DODAG cal que rebi missatges DIO d'un o més veïns. Per forçar l'enviament de DIOs, els nodes poden generar missatges DIS (DODAG *Information Solicitation*). Quan un node rep un missatge DIS, respon enviant un DIO al node que ha enviat el DIS per a que pugui unir-se al DODAG corresponent.

3.1.2. Moviment dels nodes

El DODAG també es pot veure afectat si els nodes es mouen.

Un node només canvia de "pare preferent" en dos casos: 1) si perd la connexió amb el seu "pare preferent", o bé 2) si canviant de "pare preferent" millora el *rank* del node.

Per canviar de "pare preferent" el node ha d'enviar un missatge DIS per forçar que el nou pare li enviï el missatge DIO corresponent. Si el node que canvia de "pare preferent" continua tenint connectivitat amb el "pare preferent" que tenia abans, li ha de comunicar el canvi. A més, el node que ha canviat de branca ha d'informar als seus fills per tal que els fills puguin decidir quedar-se amb ell o

triar un nou “pare preferent”. Els canvis que es produeixen a l’arbre també s’ha de propagar cap a l’arrel.

Si un node, al moure’s, perd la connectivitat amb el seu “pare preferent” i no es capaç d’unir-se a un nou “pare preferent” s’intenta solucionar el problema activant un mecanisme que es coneix com a reparació (que pot ser local o global).

3.1.3. Reparació local i global

Quan un node de l’arbre perd visibilitat amb algun veí, s’intenta restablir l’enllaç afectat. Si no és possible, cal realitzar una reconstrucció de l’arbre. El protocol RPL defineix dues maneres de reparar l’arbre: reparació local (només afecta uns quants nodes) i reparació global (afecta a tots els nodes).

La reparació global l’inicia el node arrel, i funciona com si es construís el DODAG per primera vegada.

La reparació local consisteix en intentar reconstruir només una part de l’arbre. Així, quan un node perd la connectivitat amb el seu “pare preferent” intenta unir-se a un node del mateix nivell que tenia el “pare preferent” que s’ha perdut. Si no hi ha cap node que compleixi aquest requisit, el node afectat eleva el seu *rank* a infinit (per evitar bucles) i actua com si fos un nou node que es vol unir al DODAG i envia missatges DIS als veïns esperant rebre missatges DIO com a resposta.

Una de les coses més importants a fer per prevenir cicles (*loops*) és elevar el *rank* propi fins a infinit després de fer la reparació local. En elevar el *rank* a infinit s’estalvien problemes amb el *rank* del pare. Si no s’eleva el *rank* del fill a infinit, es pot donar el cas on el *rank* del pare és més gran que el *rank* del fill actual, fet que comporta que el pare s’ajunti amb el seu propi fill provocant així un cicle.

La **Fig. 3.3** il·lustra el funcionament del mecanisme de reparació local (*local repair*).

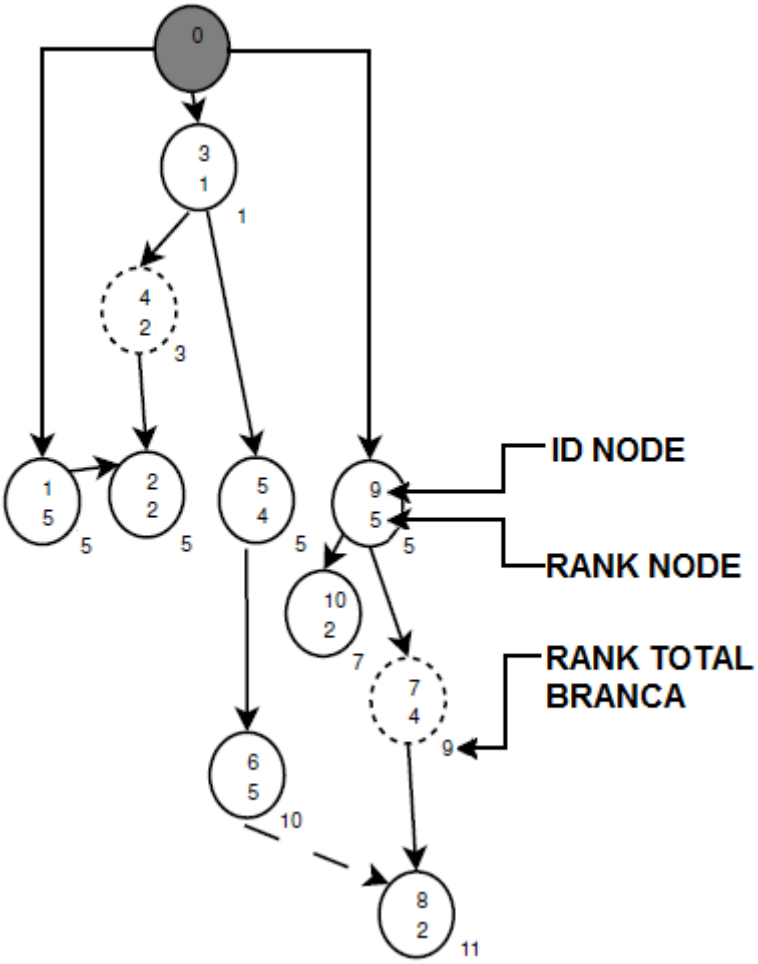


Fig. 3.3: Local repair.

3.1.4. Detecció de cicles

Un dels problemes que cal evitar en la formació del DODAG és la creació de cicles (*loops*). El mecanisme de selecció del "pare preferent" en funció del *rank* ja preveu la creació de cicles en el procés normal de formació del DODAG. No obstant això. Un moviment excessiu dels nodes de l'arbre sí que pot desembocar en la creació de cicles. L'existència de cicles es pot detectar analitzant el *rank* que sempre s'inclou en els paquets RPL. Per exemple, si un node rep un paquet que té l'etiqueta de moviment i, aquest node, té un *rank* inferior al del receptor, llavors, el node receptor conclou que el paquet no ha progressat cap amunt de l'arbre i, per tant, és inconsistent. En cas que l'arbre sigui inconsistent, s'enviarà un missatge d'error i s'iniciarà una de les reparacions explicades a l'apartat anterior.

CAPÍTOL 4. EL SIMULADOR COOJA

Contiki OS **[4]** és un sistema operatiu escrit en llenguatge de programació C **[5]**, de codi obert i desenvolupat per a petits sistemes: des d'ordinadors de 8 bits fins a sistemes integrats sobre microcontroladors, incloent nodes de xarxes de sensors. Una configuració típica de Contiki consta de 2 KB de RAM i 40 KB de ROM. Inclou una pila TCP/IP lleugera i la pila Rime (que està dissenyada especialment per a comunicacions sense fils de baixa potència) i compta amb un ampli rang de primitives de comunicació. També suporta IPv6 i protocols com RPL i 6LoWPAN.

Així doncs, Contiki permet que petits sistemes de baixa potència i mantinguts per bateries es connectin a Internet i s'utilitza per a una gran varietat d'aplicacions, com, per exemple, la monitorització de sistemes de reg, de llum, de semàfors, de soroll ambiental, etc.

Instant Contiki és una màquina virtual amb OS Ubuntu Linux que s'executa amb l'VMWare Player **[6]** i que té instal·lat el Contiki i totes les eines de desenvolupament, compiladors i simuladors utilitzats en el desenvolupament del Contiki, entre ells el simulador de xarxa Cooja.

Cooja **[7]** és un simulador escrit en Java que permet simular tant petites com grans xarxes de dispositius (*motes*). Està dissenyat específicament per simular motes amb el OS Contiki però també permet simular nodes que utilitzin un altre codi. Els *motes* es poden emular a nivell hardware (més lent però proporcionant una visió més precisa del comportament del sistema) o a un nivell menys detallat (més ràpid i convenient per a simular xarxes grans).

Una de les avantatges del Cooja és la seva flexibilitat per incorporar interfícies i *plugins* que permeten ampliar les seves funcionalitats. En aquest treball s'ha utilitzat un *plugin* de mobilitat que s'explica a l'apartat 4.4.

4.1. El mòdul Tmote Sky

Els nodes de les xarxes de sensors s'anomenen *motes* i acostumen a ser dispositius de molt baix consum, capaços de recollir, processar i enviar informació via radio a d'altres nodes de la xarxa. Les característiques de cada *mote* depenen dels seus components. Els més comuns són MicaZ mote **[8]**, Wismote **[9]**, Z1 mote **[10]** i Tmote Sky **[11]**.

En aquest treball es simulen *motes* del tipus Tmote Sky. El Tmote Sky és un mòdul sense fils d'ultra baixa potència que es va desenvolupar originalment a la Universitat de Berkeley i que està basat en l'estàndard IEEE 802.15.4. Funciona amb piles AA però també té un port USB que permet connectar-lo a l'ordinador. El baix consum d'energia és possible gràcies al microcontrolador

MSP430 **[12]** de 10 KB de RAM y 48 KB de flash ROM. També té la capacitat de despertar de manera ràpida, en menys de 6 µs. El seu processador de 16 bits RSIC li permet utilitzar menys energia tant actiu com en repòs, essent possible que duri diversos anys només amb un parell de piles AA. A més, disposa d'una antena que cobreix un rang de fins a 125m en camp obert.

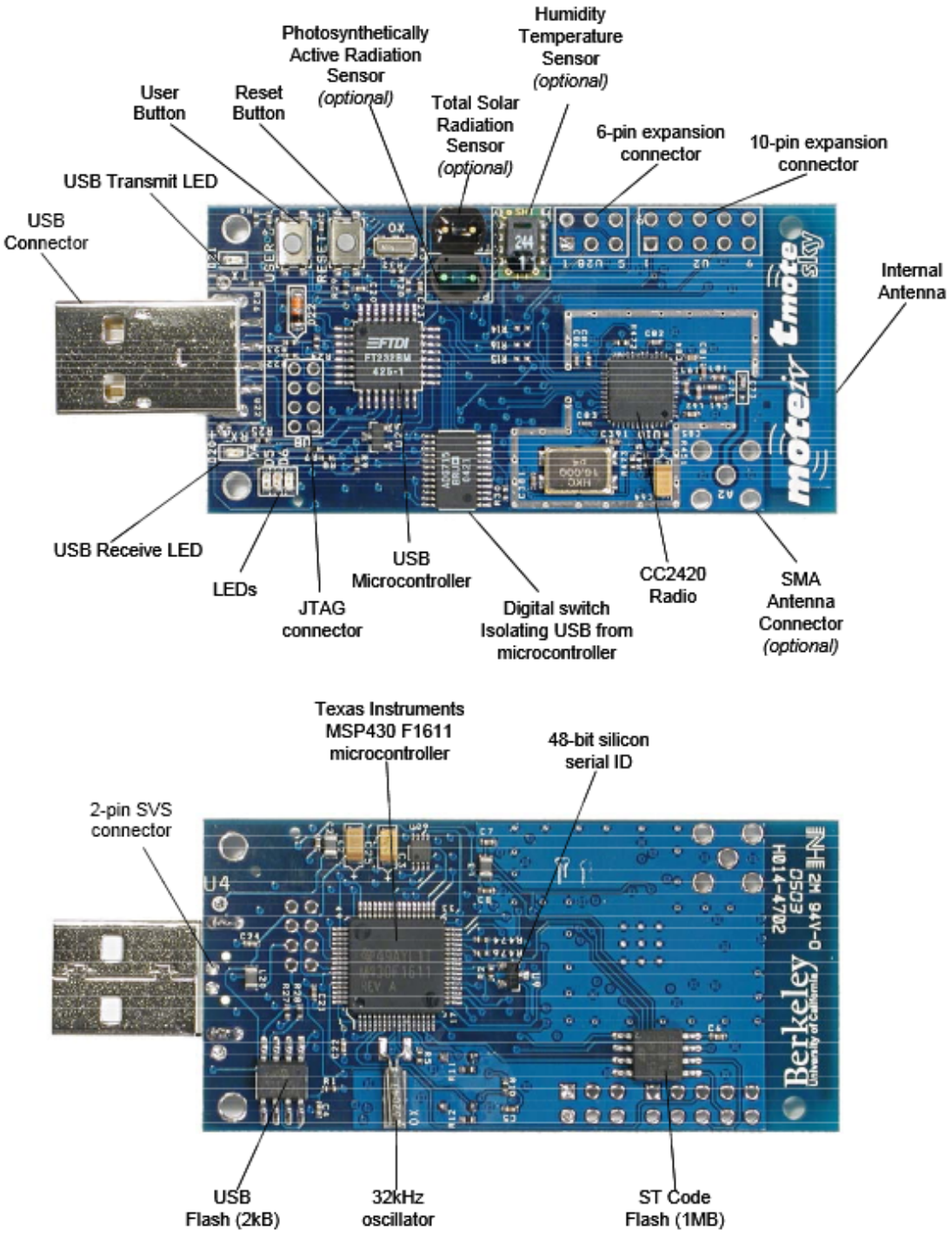


Fig. 4.4: Tmote Sky **[11]**.

La zona de cobertura, el baix consum de bateria i la capacitat de despertar ràpidament per transmetre les dades i tornar a hivernar, són les principals característiques per les qual s'ha escollit aquest tipus de *mote* per a realitzar el treball.

4.2. Exemple d'ús del simulador

A continuació es detallen els passos a seguir per a realitzar una simulació utilitzant el Cooja. L'exemple és molt senzill i consta d'un node servidor i cinc nodes client que envien paquets al servidor.

4.3.1. Descarregar el simulador

Per utilitzar el simulador Cooja cal descarregar la versió més nova de l'Instant Contiki de l'enllaç: http://www.contiki-os.org. En aquest treball s'utilitza la versió 2.6 **[13]**. L'Instant Contiki és una imatge d'una màquina virtual Ubuntu que té instal·lades les eines de desenvolupament, compiladors i simuladors utilitzats en el desenvolupament del Contiki, entre ells el Cooja.

Per arrencar la màquina virtual s'utilitza el VMWare player **[6]**.

4.3.2. Arrencada del simulador

Per arrencar el simulador, es pot utilitzat la icona de l'escriptori o executar-lo des del terminal amb les comandes:

```
cd /home/user/contiki/tools/cooja/
ant run
```

Un cop dins l'entorn gràfic del simulador, es crea una nova simulació des del menú Simulation → New i es trien els paràmetres per a la simulació (**Fig. 4.5**).

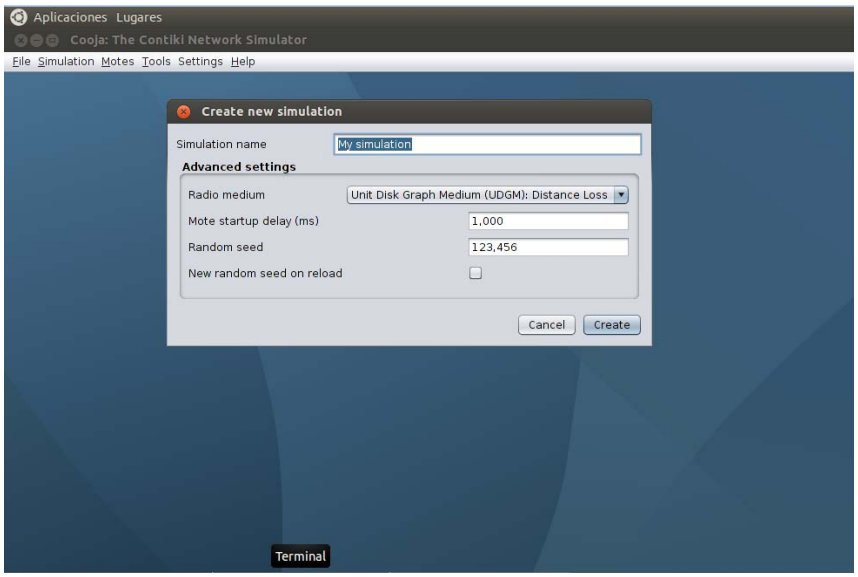


Fig. 4.5: Crear simulació.

4.3.3. Creació dels nodes

Per crear els nodes de l'escenari cal escollir el tipus de *mote* **(Fig. 4.6)** i indicar el fitxer que conté el codi que s'executarà al *mote* **(Fig. 4.7)**

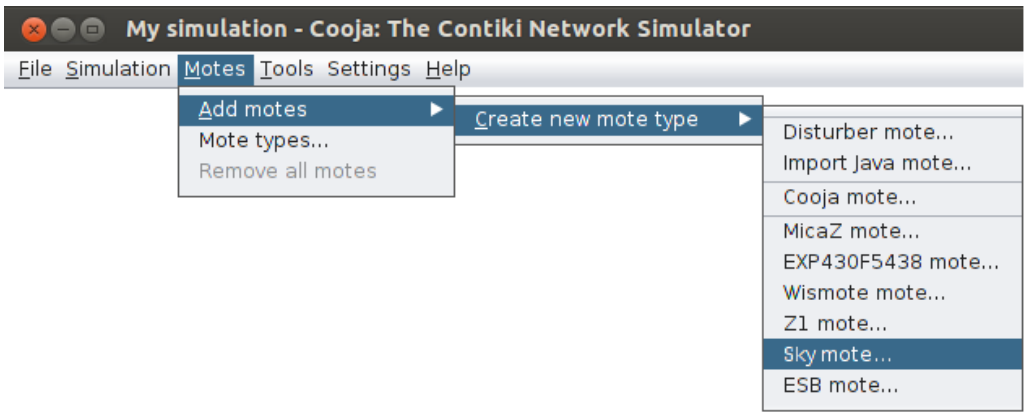


Fig. 4.6: Crear motes.

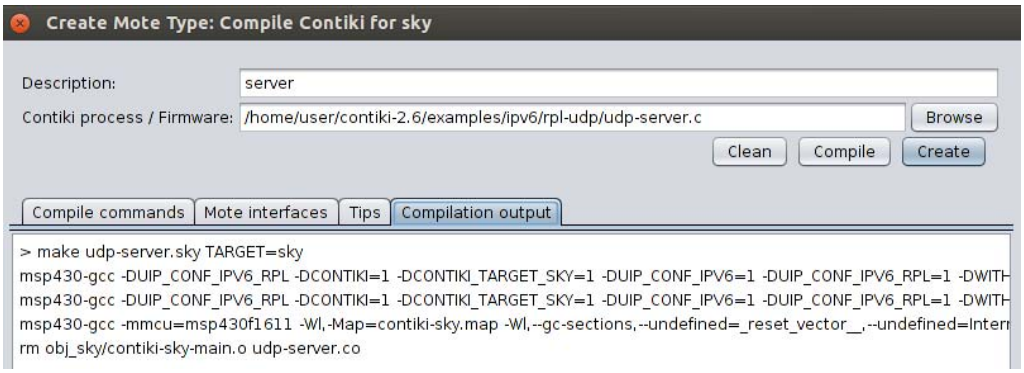


Fig. 4.7: Compilar motes.

Una vegada creat un determinat tipus de *mote* i compilat el codi que s'hi executarà, cal indicar quants *motes* d'aquell tipus es volen crear i on es volen posicionar a l'escenari **(Fig. 4.8)**.

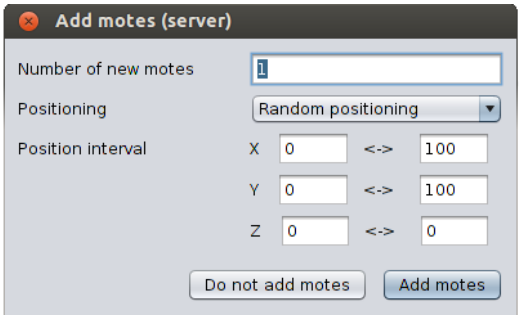


Fig. 4.8: Afegir motes

En aquest exemple s'utilitzen *motes* tipus Sky (tant pel servidor com pels clients) i el codi que s'executarà està als fitxers `udp-server.c` (per al servidor) i `udp-client.c` (per als clients).

Després d'afegir el servidor i els clients, l'aspecte de l'escenari de simulació (*network*) és el que es mostra a la **Fig. 4.9**.

Per defecte només es mostra l'ID de cada *mote* però es pot activar la visualització d'altres paràmetres a través del menú *view*. En l'exemple de la **Fig. 4.10** hi ha activades les opcions de visualització del trànsit entre nodes, el tipus de *mote*, els LEDs, el radi de transmissió i la quadricula de l'escenari.

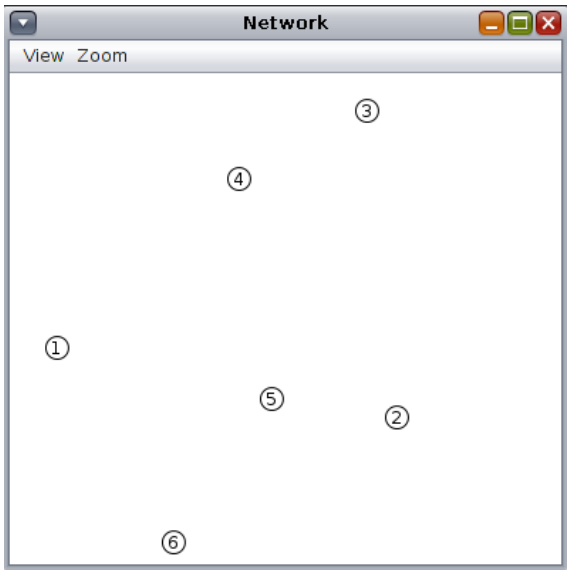


Fig. 4.9: Aspecte inicial de l'escenari de simulació.

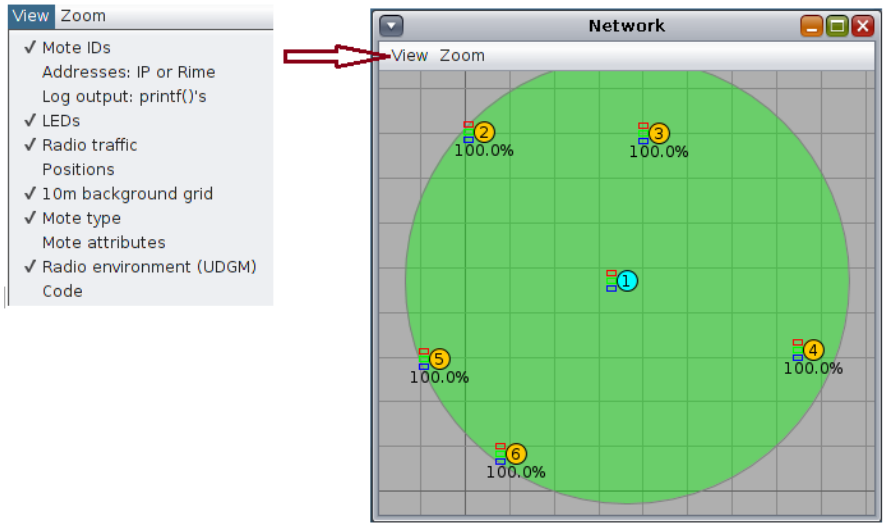


Fig. 4.10: *Network* aspecte final.

4.3.4. Execució de la simulació

Una vegada preparat l'escenari, és moment d'iniciar la simulació prement el botó START.

Durant l'execució de la simulació es pot interactuar amb l'aplicació, modificar la velocitat de simulació, canviar les característiques dels *motes* simulats (rang de transmissió, valors de les trames, estat dels nodes), canviar les opcions de visualització del simulador, etc.

En aquest exemple, les dades que s'intercanvien els nodes a l'inici de la simulació corresponen als paquets DIO i DAO del protocol RPL que els nodes

envien per tal de crear el DODAG i omplir les taules d'encaminament. Aquest intercanvi de paquets és el que mostren les fletxes blaves de la finestra *network* i els rectangles blaus i verds de la finestra *timeline* de la **Fig. 4.11**.

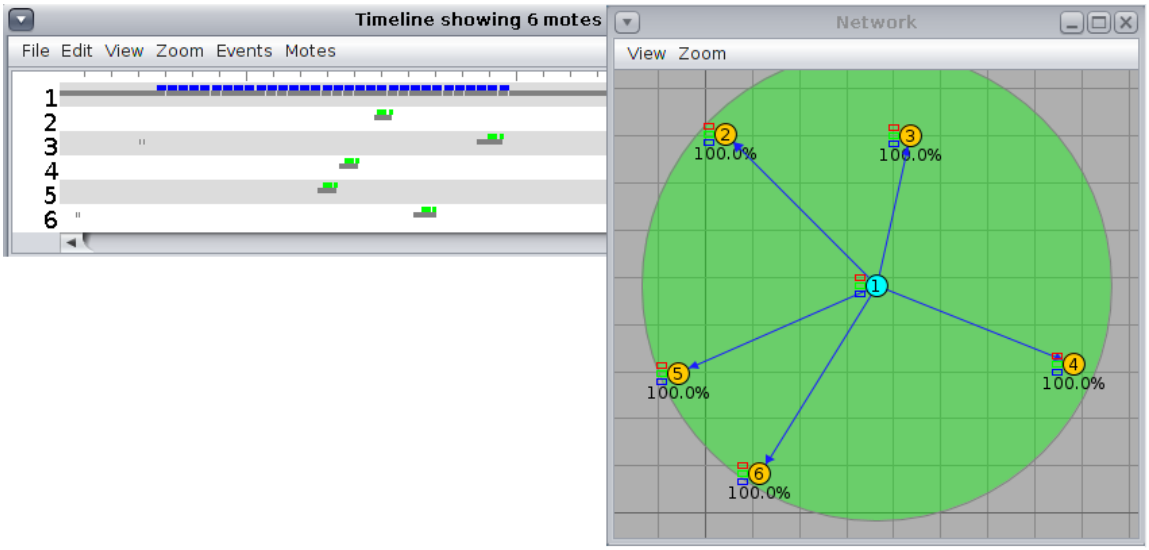


Fig. 4.11: Intercanvi de paquets.

A partir del minut 1 de la simulació, els nodes client comencen a enviar els paquets de dades amb el missatge "*Hello*" cap al node servidor. La **Fig. 4.12** mostra la transmissió d'un d'aquests paquets enviat pel node 5 i rebut pel node 1.

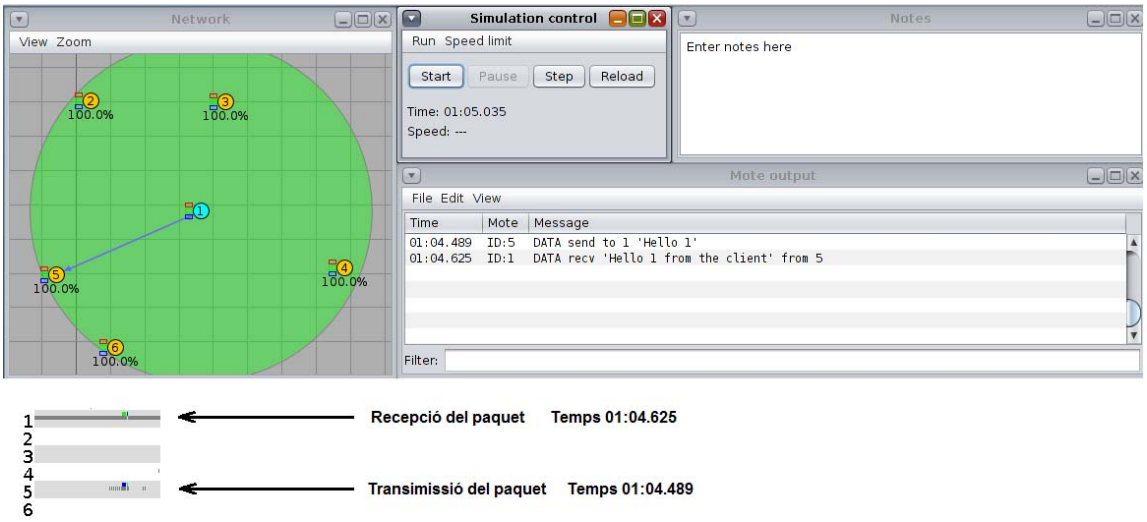


Fig. 4.12: Transmissió de les dades.

4.3. Mobilitat

El simulador Cooja disposa d'un conjunt d'eines i mòduls addicional que permeten dotar el simulador de funcionalitats avançades o simular aplicacions amb unes característiques determinades. Una d'aquestes eines permet definir un patró de moviment dels nodes de l'escenari. En aquest treball es desitja que els nodes tinguin un cert moviment dins l'escenari i per tant és necessari instal·lar-la.

4.4.1. Instal·lació de l'eina de mobilitat

Per instal·lar l'eina cal anar al directori `/home/contikiprojects/sics.se` i copiar la carpeta *mobility* al directori `contiki-2.X/tools/cooja/apps/`.

Després, des del Cooja, cal activa-la des del menú *Seetings* → Cooja *extensions* **(Fig. 4.13)**.

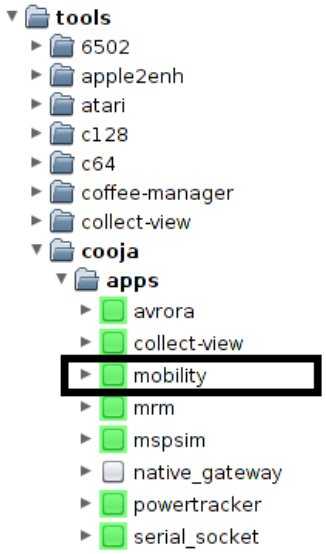


Fig. 4.13: Activar l'eina de mobilitat.

4.4.2. Funcionament de la mobilitat

El que fa l'eina de mobilitat és carregar un fitxer de text on hi ha descrit el moviment dels nodes seguint el format que s'indica a la **Fig. 4.14:**

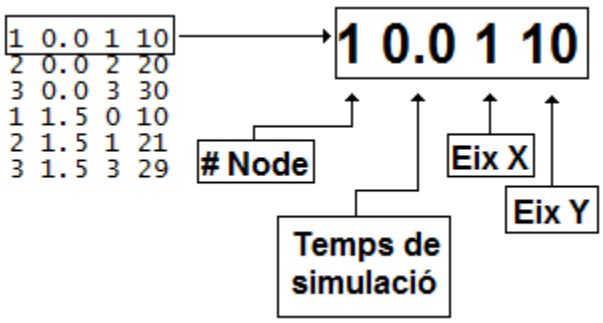


Fig. 4.14: Arxiu de mobilitat.

Cada línia del fitxer està format per quatre columnes. La primera columna indica el node al qual es refereix aquella fila. La segona columna indica l'instant de simulació en la qual el node assolirà la posició indicada pels valors de les columnes 3 i 4. La columna 3 indica la posició a l'eix X (en metres) i la columna 4 indica la posició a l'eix Y (en metres). També és possible definir un tercer eix (eix Z) per descriure un moviment en tres dimensions.

CAPÍTOL 5. DISSENY I SIMULACIÓ DEL SISTEMA

En aquest capítol es descriu el procés que s'ha seguit per a poder simular el sistema proposat utilitzant el Cooja.

Per a poder simular el sistema cal:

- Escollir el tipus de nodes que s'utilitzaran per a representar tant els nodes mòbils (animals) com els nodes fixos del sistema i definir la posició dels nodes fixos.

- Definir el patró de moviment dels nodes mòbils.

- Implementar les funcions que permetin enviar l'identificador de cada node al node recol·lector i representar gràficament la informació emmagatzemada al node recol·lector.

5.1. Selecció dels nodes i definició de l'escenari

Cada node del simulador Cooja executa el Contiki amb el codi específic que s'indica en la seva creació. En aquest treball s'utilitzen *motes* tipus Sky que simulen mòduls Tmote Sky. El codi que s'ha utilitzat com a base per als nodes arrel d'un DODAG és el que hi ha al fitxer *`udp-sink.c.`* del directori `examples/ipv6/rpl-collect/`. Per als nodes mòbils i pels fixos que no actuen d'arrel de cap arbre RPL s'ha utilitzat com a base el fitxer *`udp-sender.c`* que hi ha al mateix directori. Aquests fitxers inclouen el codi que permet comunicar els nodes entre sí i enviar les dades necessàries cap al node arrel.

Per definir la quantitat i la posició dels nodes arrel (*sinks*) i dels nodes fixos que no actuen d'arrel (*senders*) s'han realitzat diferents proves:

1. Disseny amb un únic node *sink*.

   Utilitzant un únic node fix que actuï d'arrel de l'arbre (tal i com mostra la **Fig. 5.15**) la probabilitat que els nodes mòbils estiguin desconnectats de l'arbre (i per tant, no sigui possible localitzar-los) és molt elevada.

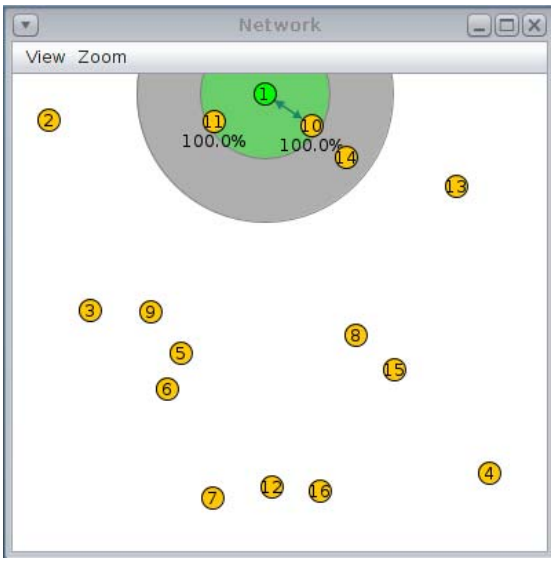


Fig. 5.15: Simulació proposada.

A la **Fig. 5.15**, el node verd és l'arrel de l'arbre (i el recol·lector de les dades) i els nodes grocs representen els dispositius mòbils (animals). Com s'observa a la figura, només els nodes 10 i 11 (per visió directa) i el node 14 (a través del node 10) poden comunicar-se amb l'arrel. La resta de nodes no es poden comunicar amb el node recol·lector i, per tant, no es pot representar la seva posició.

2. Disseny amb un únic node *sink* i cobertura completa per nodes *sender* fixos.

La **Fig. 5.16.** mostra un escenari on hi ha un únic node *sink* (en color verd) i suficients nodes fixos (en color groc) per cobrir tota la zona de pastura dels nodes mòbils (en color fúcsia).

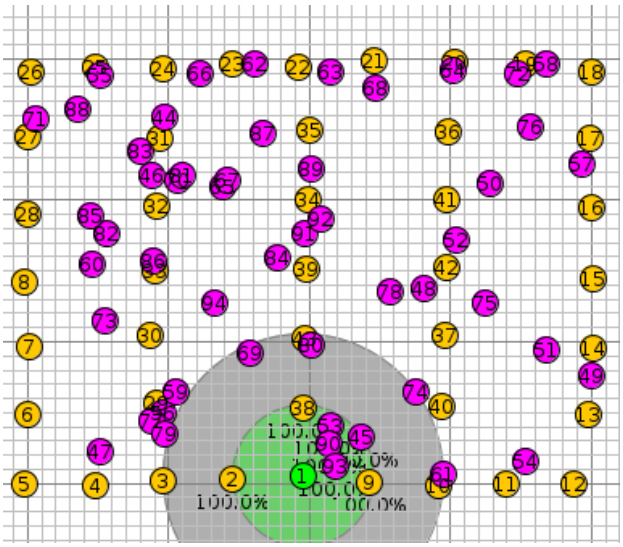


Fig. 5.16: Simulació proposada.

El problema d'aquesta configuració és l'elevat nombre de nodes (entre fixos i mòbils) per a un únic node arrel. La implementació actual del protocol RPL del Contiki no suporta el mode *non-storing*. Per tant, tots els nodes intermedis de l'arbre han de mantenir una taula d'encaminament amb les rutes per accedir a tots els nodes que pengen de la seva branca.

La configuració per defecte del protocol RPL del Contiki defineix un màxim de 20 veïns i 20 rutes a la taula d'encaminament (`/contiki/paltform/sky/contiki-conf.h`) i en aquest escenari és molt probable que un node tingui més de 20 descendents a la seva branca. A més, si es canvia el valor per defecte del fitxer de configuració per intentar guardar més entrades a la taula d'encaminament es posa de manifest el problema de la memòria limitada dels *motes*. D'aquest problema de memòria se'n parla i s'explica el que s'ha fet per intentar ampliar-la el màxim possible a l'apartat 5.3.

3. Disseny amb un únic node *sink* i, nodes fixos col·locats estratègicament.

Finalment es proposa utilitzar un únic node arrel (*sink)* i diversos nodes fixos (*senders*) situats estratègicament dins la zona de pastura, tal i com mostra la **Fig. 5.17** (on el node arrel és el de color verd i els nodes fixos són els grocs).

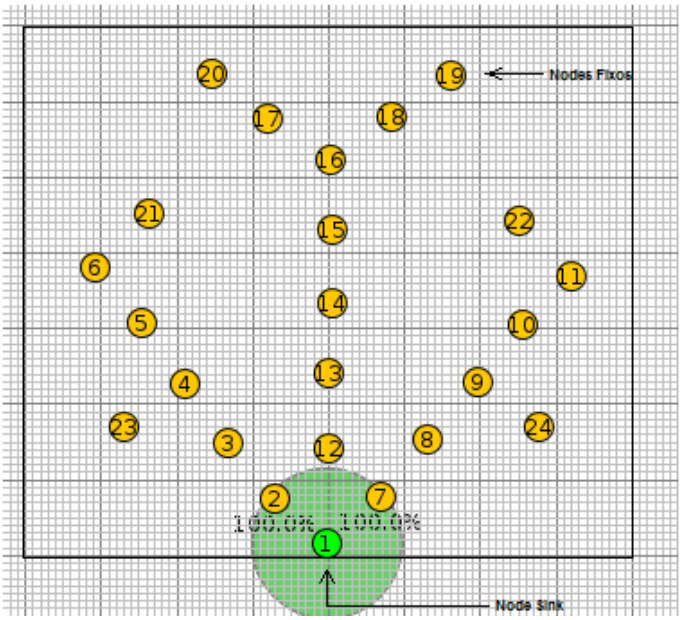


Fig. 5.17: Simulació proposada.

Aquest disseny permet cobrir gran part de la zona de pastura (no tota) amb un nombre més reduït de nodes fixos que el proposat en l'escenari anterior i és el que s'ha utilitzat per a fer les simulacions en aquest treball. No obstant, cal tenir present que en aquest escenari el nombre

de nodes també estarà limitat per la memòria dels dispositius, ja que, com s'ha comentat anteriorment, la implementació de l'RPL del Contiki no disposa de la facilitat de *non-storing* i la memòria dels *motes* és molt petita.

Cal mencionar que l'escenari ideal per al sistema que es proposa en aquest treball estaria format per diversos nodes arrel (i, per tant, diversos arbres) que cobrissin tota la zona de pastura. Com que cada arbre només s'hauria d'encarregar d'una determinada zona, el nombre de nodes de l'arbre seria, en teoria, més reduït que en l'últim escenari proposat i, per tant, el nombre de rutes que hauria d'emmagatzemar cada node seria menor. Cada arbre tindria una arrel i diversos nodes fixos que en constituirien l'estructura base sobre la qual s'anirien afegint els nodes mòbils (animals). Cada node arrel hauria d'enviar la informació relativa al seu arbre cap al node recol·lector, que s'encarregaria de processar tota la informació rebuda i generar la representació gràfica de la posició dels animals dins la zona de pastura. Idealment, els nodes arrel serien nodes més complexes que els *motes* que actuen de nodes fixos i mòbils, estarien connectats permanentment a la corrent elèctrica i es podrien comunicar amb el node recol·lector a través d'una xarxa (fixa o sense fils) convencional. En aquest treball no s'ha realitzat la programació i simulació d'aquest tipus de nodes, deixant aquesta opció (que és una ampliació natural de l'escenari proposat) per a treballs futurs.

5.2. Definició del patró de moviment dels nodes mòbils

Per simular el moviment dels animals s'utilitza l'eina de mobilitat que s'ha comentat al capítol anterior. No obstant, per crear el fitxer de configuració que l'eina utilitza per determinar el moviment dels nodes no s'ha utilitzat cap del patrons de mobilitat disponible **[14]** ja que cap d'ells permet emular el moviment dels animals de forma realista. Per tant, el que s'ha fet és programar una aplicació en Microsoft Visual C# Sharp 2010 **[15]** que genera el fitxer.

Per definir el moviment dels animals s'ha tingut en compte que les vaques estan parades un 80% del temps i que, quan es mouen, recorren poca distància (uns 2,5 metres, de mitjana). A més, en el 95% dels casos no giren 180º si no han estat parades una estona abans.

El programa dissenyat **(Fig. 5.18)** permet definir el número de nodes mòbils, l'identificador del primer node mòbil (els nodes amb identificadors inferiors a l'indicat són fixos), la durada de la simulació (en segons), la velocitat mitjana dels nodes mòbils (en metres/segon) i les dimensions de terreny on es mouen els nodes. Com a sortida, el programa genera un fitxer en el format que requereix l'eina de mobilitat del Cooja **(Fig. 5.19)**.

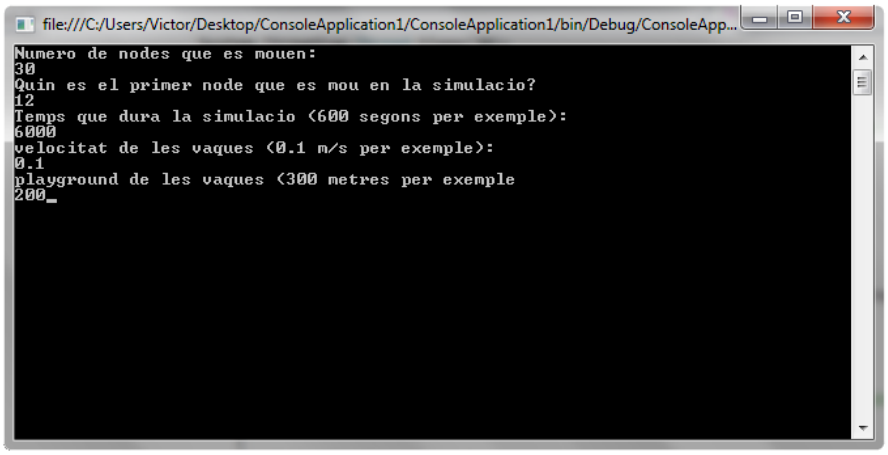


Fig. 5.18: Aplicació de mobilitat.

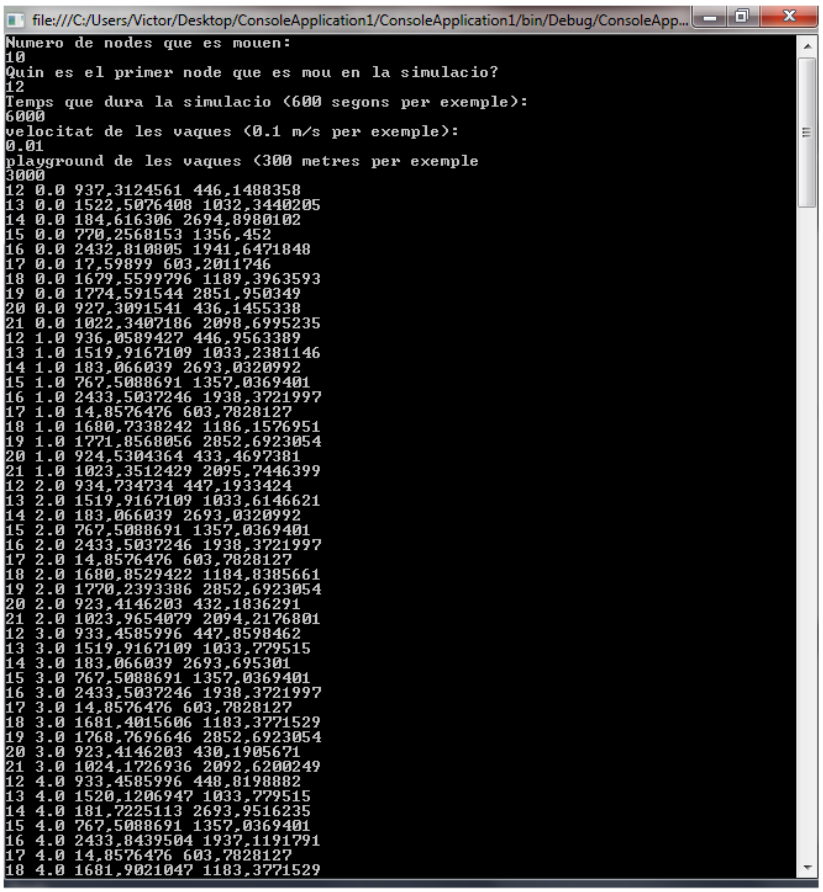


Fig. 5.19: Aplicació de mobilitat creant l'arxiu.

5.3. Enviament i representació de les dades

L'objectiu d'aquest treball és aconseguir que els nodes enviïn informació de la seva localització a l'arrel per tal que l'arrel representi la localització dels nodes a l'escenari.

Per poder enviar les dades al node arrel s'ha hagut d'ampliar al màxim la memòria dels *motes* per maximitzar el nombre de rutes que poden emmagatzemar i s'han hagut de modificar algunes de les funcions dels fitxers `sink.c` i `sender.c` del directori `/examples/ipv6/rpl-collect/`.

Finalment, s'ha creat un script al node arrel per tal de representar el fitxer que descriu la relació entre els nodes de l'escenari utilitzant l'aplicació Graphviz **[16].**

5.3.1. Millores en l'ús de la memòria

En aquest apartat s'expliquen els canvis realitzats per tal d'ampliar al màxim possible la memòria Flash i la RAM dels *motes*.

La capacitat de memòria del microcontrolador MSP430 és de 48 KB de Flash i 10 KB de RAM **[17]**.

Observant una compilació amb tots els paràmetres per defecte del Contiki OS, s'obtenen els següents valors:

- El node *sink* utilitza 44.4 KB de memòria Flash i 9.2 KB de memòria RAM.
- El node *sender* utilitza 46.5 KB de memòria Flash i 9.3 KB de RAM.

Disposant de tan poca memòria resulta molt difícil poder implementar cap de les funcions que cal programar. Per tant, es decideix eliminar totes aquelles funcionalitats que consumeixen memòria i que no són necessàries per a acomplir l'objectiu d'aquest treball.

El primer pas és eliminar el TCP de la compilació, ja que tot l'enviament de missatges es fa via UDP. Per eliminar el TCP de la compilació, cal anar a l'arxiu `contiki/core/net/uipopt.h` i posar un 0 a l'opció `UIP_CONF_TCP`:

```
#define UIP_CONF_TCP 0.
```

Després de fer això, la memòria queda:

- El node *sink* utilitza 40.5 KB de memòria Flash i 9.0 KB de memòria RAM.
- El node *sender* utilitza 42.6 de memòria Flash i 9.0 KB de memòria RAM.

El segon pas és optimitzar la compilació dels nodes modificant alguns *flags* de compilació a l'arxiu `Makefile.include` del directori arrel `/home/contiki/`:

```
CFLAGS += -ffunction-sections
LDFLAGS += -Wl,--gc-sections,--undefined=_reset_vector__,
--undefined=InterruptVectors,--undefined=_copy_data_init__,
--undefined=_clear_bss_init__,--undefined=_end_of_init__
```

Després de fer això, la memòria queda d'aquesta manera:

- El node *sink* utilitza 38.1 KB de memòria Flash i 9.0 KB de memòria RAM.
- El node *sender* utilitza 39.7 de memòria Flash i 9.0 KB de memòria RAM.

En aquest punt ja es pot començar a treballar en la programació de les funcions d'enviament i recepció de dades, però per tenir més marge es modifica la configuració del *mote* Sky al fitxer `/contiki/paltform/sky/contiki-conf.h` per reduir el nombre d'entrades de la taula d'encaminament (de 30 a 15) i el nombre de veïns que es poden guardar a la *cache* (de 30 a 10).

```
UIP_CONF_DS6_NBR_NBU = 10
UIP_CONF_DS6_ROUTE_NBU = 15
```

5.3.2. Programació dels nodes *sink* i *sender*

El que s'ha fet en aquest treball és programar el node *sender* de manera que enviï la informació cap al node *sink*, i aquest, la processi.

El primer que fa el node *sender* abans de preparar el paquet és recórrer la taula de veïns que té i agafar les direccions IPv6 corresponents a tots els veïns que té en estat REACHABLE. Tots els nodes que té dins del seu radi de visió apareixen en l'estat REACHABLE, mentre que els nodes que han sigut veïns d'aquest però ara no estan dins del radi de visió, apareixen en estat STALE. Si la taula dels veïns s'omple (màxim 10 veïns després de configurar-la), cada veí nou que arriba substitueix el node més antic en estat STALE, i en cas que hi hagi més de 10 veïns al voltant, hi ha un mecanisme propi del sistema Contiki que defineix una aleatorietat a l'hora d'agafar els veïns. Per tant, si hi ha més de 10 veïns, entre tots s'enviarà tota la informació al node *sink.*

Com que el sistema està molt limitat a causa de la memòria, per estalviar informació innecessària, es programa el node *sender* perquè agafi les direccions IPv6 i només es quedi amb l'identificador del node que té com a veí. D'aquesta manera s'estalvia molts bits a l'hora de fer l'enviament. Així doncs, el paquet que s'envia al node *sink* és un paquet format per una estructura amb la direcció del node que l'envia i els identificadors de tots els veïns que veu en aquest moment.

D'altra banda, el node *sink* rep el paquet que són enviats per tots els *senders*, i els processa. El node *sink* està programat de manera que en rebre un paquet, processi les dades i les guardi en una taula pròpia per després poder fer la representació gràfica del sistema.

Quan rep un paquet, agafa la direcció IPv6 del *sender* i es queda amb l'identificador. Una vegada coneix el node emissor, busca a la seva pròpia taula per si ja tenia coneixement previ d'aquest node. En cas de no conèixer el node emissor, guarda tota la informació en la seva pròpia taula i s'actualitza la informació per a fer la representació. En cas que el node emissor ja sigui conegut, compara la informació interna del paquet rebut i la informació anterior que tenia en la seva taula. D'aquesta manera, comprova els canvis que hi ha en la informació rebuda, modifica les dades de la seva pròpia taula i prepara la informació per a fer la representació. Així doncs, cada vegada que arriba un paquet d'un node emissor i la informació és diferent a la que el node *sink* tenia fins aleshores, es produeix una actualització de la taula i una modificació en l'arxiu a representar.

El node *sink* és el node encarregat de preparar totes les dades per a la seva futura representació. El *sink* no es pot programar de manera que ho automatitzi les funcions de guardar dades en un arxiu a part i representar el gràfic a causa de la poca memòria de que disposa. Per fer això s'han de programar funcions externes al codi del *sink* que puguin guardar la informació en un arxiu apart i representar el gràfic. Per poder fer això es programaran les funcions en scripts que pot fer servir el Cooja **[18]** i en scripts de sistema (*Shell scripts* **[19]**).

5.3.3. Representació de l'arbre arrel

La representació de l'arbre arrel es fa amb totes les dades recopilades en el node *sink.*

Per fer la representació, el primer que es fa és escriure en un fitxer les dades del *sink* que han estat preparades anteriorment dins del codi. La manera triada per guardar les dades en un arxiu és mitjançant un *script* del Cooja **[18]**. Aquest *script*, reconeix els canvis en la taula pròpia del *sink,* i cada vegada que hi ha un canvi en aquesta, obre l'arxiu, reescriu la informació de la taula del *sink* i tanca l'arxiu. D'aquesta manera, cada vegada que el *sink* actualitza la taula amb les noves dades, el *script* del Cooja les guarda en un arxiu ja preparat perquè el graphviz pugui representar les dades en un gràfic.

Tot seguit s'ensenya el codi utilitzat:

```
//Llibreries
importPackage(java.io);

// Utilitzem JavaScript objects
outputs = new Object();

//OUTPUT TEXT, L'arxiu del log està en la carpeta
//home/contiki/tools/cooja/build

while (true) {
      //Si es cert, es crea l'arxiu
      if (!outputs[id.toString()]) {
            // S'obre l'arxiu per a escriure.
            outputs[id.toString()]=new FileWriter("log_" + id +".dot");
      }

      //Escrivim en els següents 3 if's preparant l'arxiu per al
      //graphviz
      if(msg.indexOf("########")!= -1) { //Primera línia (inici del
                                          //graphviz)
            var inici = ("digraph{ \n")
            outputs[id.toString()].write(inici);
            var linia = msg.substring(0)
            outputs[id.toString()].write(linia + "\n");
      }
      if(msg.indexOf("NODO")!= -1) {//Els nodes que es veuen
            var linia = msg.substring(5)
            outputs[id.toString()].write(linia + "\n");
      }
      if(msg.indexOf("@")!= -1) {//última línia
            var final = ("####################################\n}")
            outputs[id.toString()].write(final);
            for (var ids in outputs) {
                  outputs[ids].close();
            //Quan tenim tot escrit tanquem l'arxiu per representar
            }
            //Esperem al següent graph, fins que no arriba START no
            //inicia la següent construcció
            YIELD_THEN_WAIT_UNTIL(msg.equals("START"));
            if(msg.equals("START")==true) {
              outputs[id.toString()]=new FileWriter("log_"+id +".dot");
            }
      }
      Try { //Control d'errors
            //Mentre no hi hagi excepció, seguir.

            YIELD();
      }
      catch (e) {
            //Tanquem els arxius si salta excepció
            for (var ids in outputs) {
                  outputs[ids].close();
            }
            //forcem acabar l'script al saltar alguna excepció
            throw('test script killed');
      }
}
```

Després de preparar l'arxiu per representar es busca la manera de representar l'arxiu cada vegada que aquest canvia. Per fer això s'ha programat un *script* de sistema (*Shell script* **[19]**). La funció que té aquest *script* una vegada s'arrenca, és estar sempre en funcionament (en *background*) mentre es centra en l'arxiu que rep les modificacions. Cada vegada que rep les modificacions arrenca el Graphviz per a l'arxiu seleccionat, fa el gràfic corresponent i tanca el Graphviz. D'aquesta manera, sempre que rep una actualització representa el gràfic de manera automàtica.

El codi utilitzat és el següent:

```
#!/bin/sh

cd /home/user/contiki/tools/cooja/build

while inotifywait -e modify log_1.dot;
do dot -T png -O log_1.dot
done
```

CAPÍTOL 6. SIMULACIÓ I VERIFICACIÓ

En aquest capítol es mostra el resultat de realitzar una simulació del sistema proposat utilitzant el simulador Cooja.

6.1. Activació dels nodes fixos

El primer pas de la simulació consisteix en posicionar els nodes fixos de l'escenari. Aquests nodes estan col·locats a una distància de 100 m entre ells, i serviran com a referència per a determinar, posteriorment, la ubicació dels animals. Així doncs, s'arrenca el simulador Cooja i es reparteixen un total de 24 nodes dins una zona quadrada de 800m x 800m, que representa el camp de pastura, tal i com es mostra en la **Fig.6.20**

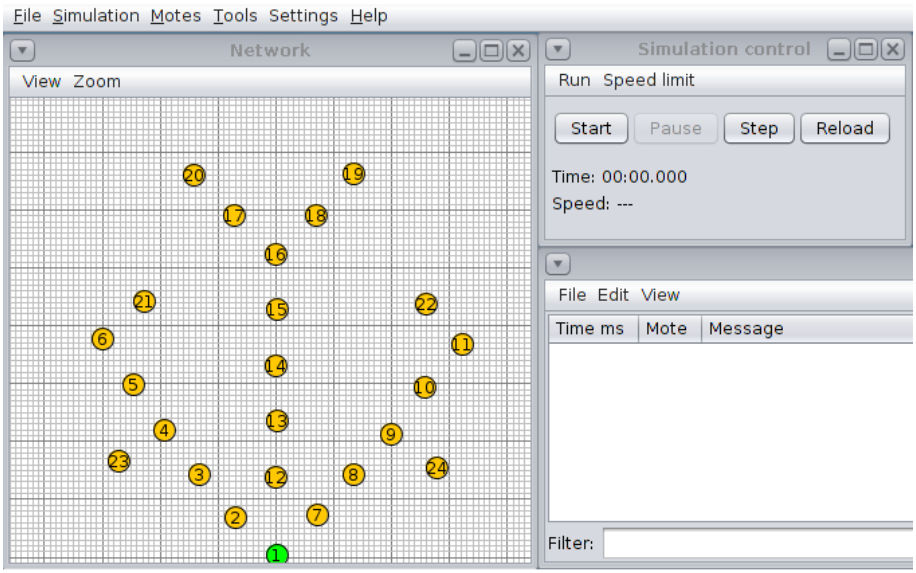


Fig. 6.20: Xarxa a muntar.

Un dels nodes fixos (node 1 a la **Fig.6.20**) està configurat per actuar com a recol·lector i analitzador de les dades. Aquest node actuarà d'arrel de l'arbre RPL i serà l'encarregat de guardar en un fitxer tota la informació que envien els nodes identificant el seus veïns. Per guardar aquesta informació en un fitxer cal activar el *script* del Cooja on hi ha programada aquesta funció, tal i com es mostra a la **Fig. 6.21**.

A partir del fitxer generat pel node arrel, es dibuixa el gràfic de l'escenari. Per fer-ho, cal arrencar (tal i com mostra la **Fig.6.21**) el programa que durant tota la simulació va comprovant els canvis que es produeixen en el fitxer esmentat anteriorment per tal de representar el gràfic que especifica la relació entre els nodes de la xarxa.

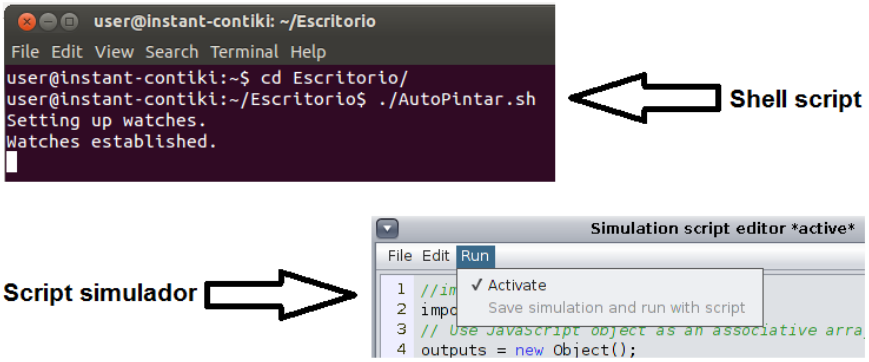


Fig. 6.21: Activar scripts.

Un cop s'inicia la simulació, els nodes comencen a enviar-se missatges DIO i missatges DAO (tal i com s'il·lustra a la **Fig.6.22**) fins a completar la construcció del DODAG arrelat al node 1.

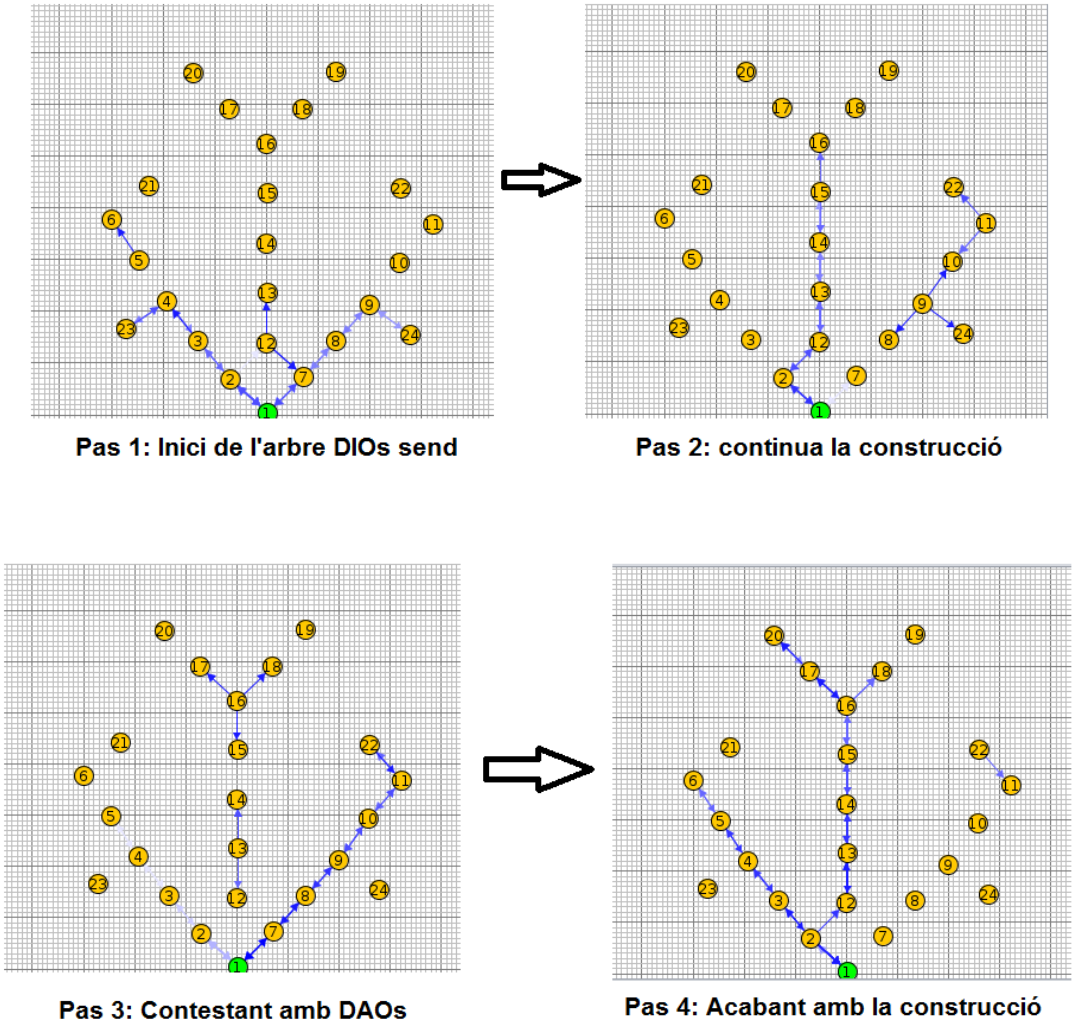


Fig. 6.22: Intercanvi de paquets per a la construcció del DODAG.

En paral·lel, el node 1 va representant el contingut del fitxer on s'emmagatzema la informació rebuda per l'arrel en relació als veïns que veu cada node. La **Fig.6.23** mostra la representació del fitxer quan encara no està construït el DODAG. Els arcs entre els nodes representen la relació de veïnatge. Així, per exemple, el node 1 té com a veïns els nodes 2 i 7. El node 7 té com a veïns els nodes 1, 12 i 8 i així successivament. La **Fig.6.24** mostra la representació final del gràfic quan s'ha completat la construcció del DODAG que conté els nodes fixos.

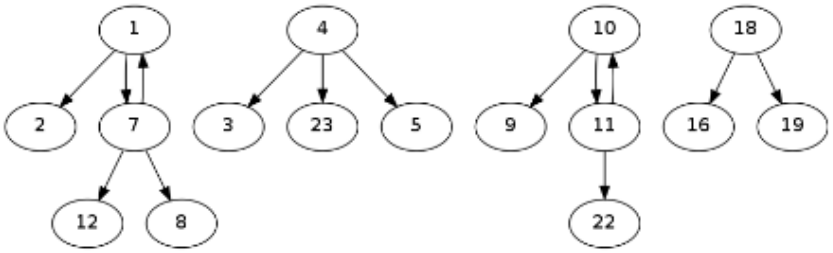


Fig. 6.23: Representació durant la construcció del DODAG.

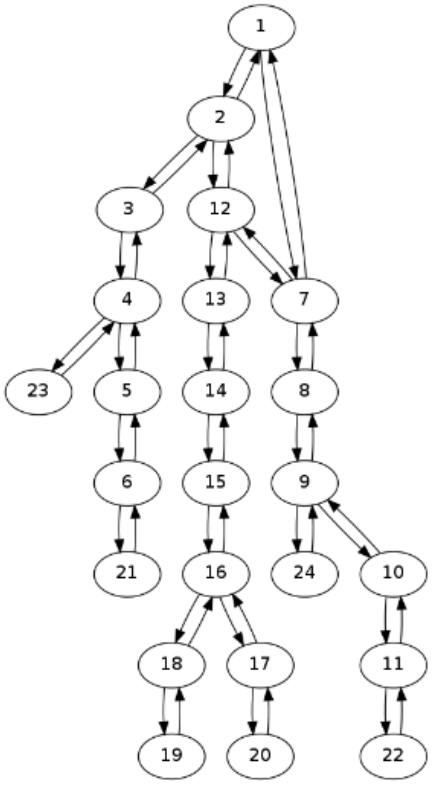


Fig. 6.24: Representació un cop finalitzada la construcció del DODAG.

6.2. Segona part de la simulació

Un cop es disposa de la representació gràfica dels nodes fixos de l'escenari, es procedeix a introduir 30 nodes mòbils que representen els animals. Per simular el moviment dels animals es crea un fitxer utilitzant el programa explicat al capítol anterior que fixa un moviment aleatori per als 30 nodes esmentats d'una duració de 50 minuts i amb una posició inicial completament aleatòria dins del camp quadrat de 800m x 800m. La **Fig.6.25** mostra l'escenari amb els nodes fixos (verd i groc) i els nodes mòbils (en fúcsia).

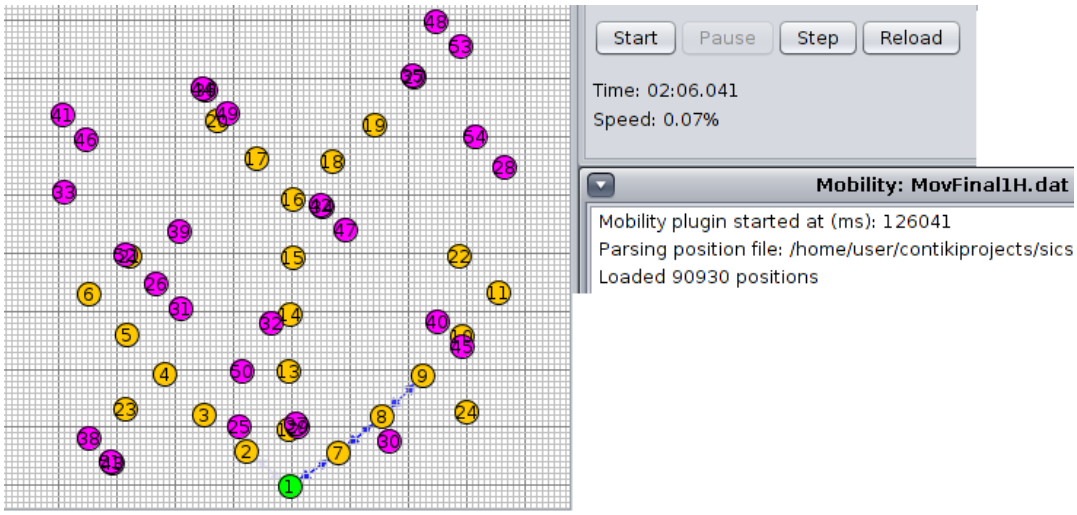
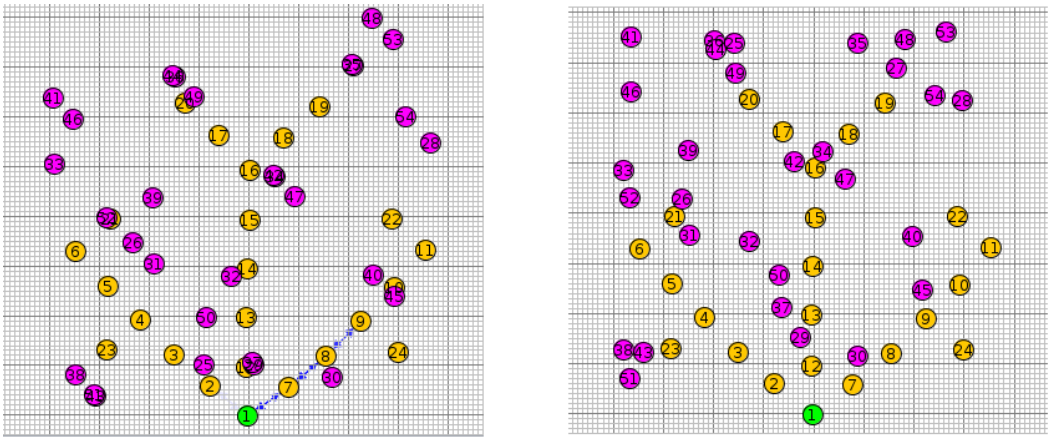


Fig. 6.25: Escenari amb nodes mòbils i fixos.

A mesura que els nodes mòbils es van unint al DODAG, van enviant la informació a l'arrel de l'arbre i es van representant al gràfic del node recol·lector. La **Fig. 6.26** il·lustra el canvi de posició dels diferents nodes durant la simulació. La **Fig. 6.27** mostra el gràfic generat pel node recol·lector al minut 10:44 de simulació.



Minut 2:04 de simulació, inici d'aquesta.

Minut 10:44 de simulació estat actual de la xarxa

Fig. 6.26: Ubicació dels nodes mòbils en diferents instants de la simulació.

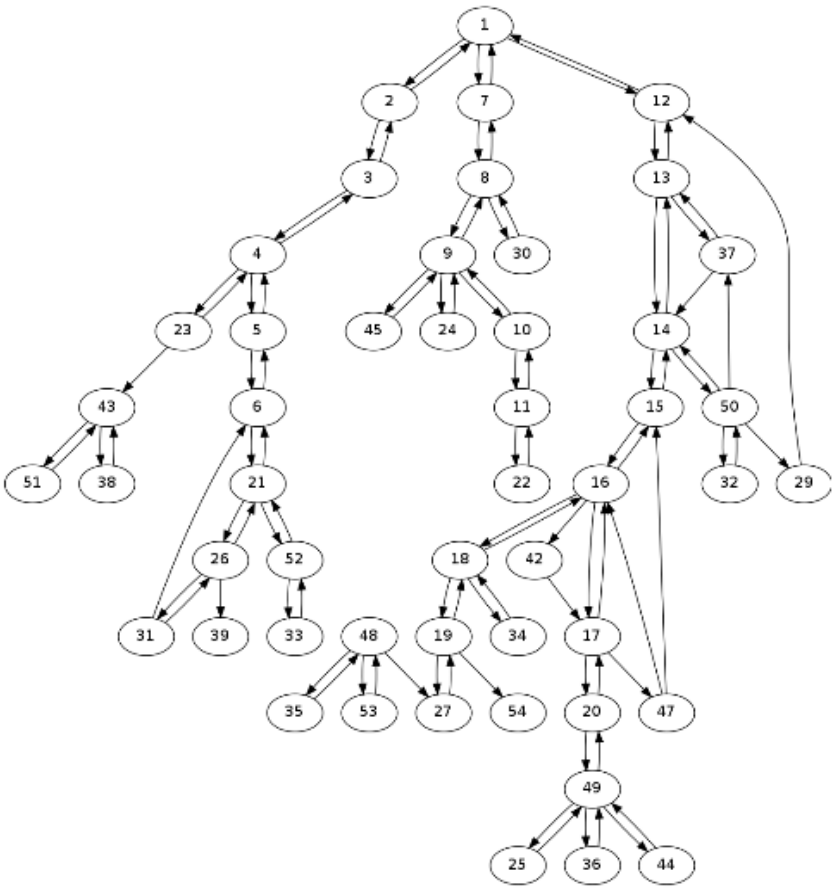


Fig. 6.27: Graf de la xarxa al minut 10:44 de simulació.

6.3. Conclusions i problemes en el sistema

Així doncs, tal i com es pot observar en la simulació explicada en aquest capítol, la feina feta en aquest treball permet dibuixar un gràfic que representa els nodes de la xarxa. Aquest gràfic es genera a partir de la informació que envien els nodes a través de l'arbre RPL. En particular, la informació que envien els nodes és el conjunt de veïns que el node veu. No obstant això, encara hi ha aspectes a completar per acabar de tenir el sistema de localització de ramat que es proposava inicialment. En aquest apartat es mencionen dues de les millores que caldria fer per completar la feina:

a) Proporcionar informació real de la posició dels nodes

L'objectiu del sistema és proporcionar al pastor una idea aproximada de la ubicació dels animals. No obstant això, el gràfic que mostra l'aplicació presentada en aquest treball no proporciona aquesta informació. Caldria modificar la representació del gràfic per tal que representés la zona de pastura i la ubicació exacte dels nodes fixos (informació, en principi, coneguda). A partir d'aquí, la posició dels animals no es podria conèixer de forma exacta (si no

s'utilitza algun sistema addicional, per exemple GPS) però es podria tenir una idea aproximada a partir de la relació de veïnatge entre els nodes mòbils i els nodes fixos.

a) Corregir els problemes en l'actualització de les taules d'encaminament dels nodes

Quan un node es desplaça és possible que hagi de canviar de "pare preferent" del DODAG i, a la vegada, és possible que provoqui que altres nodes el perdin com a "pare preferent" i hagin de buscar una alternativa per seguir units a l'arbre. Tots aquests canvis s'han de traduir en canvis a la taula d'encaminament. No obstant això, la implementació actual del Contiki no actualitza correctament les taules d'encaminament dels nodes (no esborra les rutes antigues encara que un node deixi de ser accessible. Tota la informació dels problemes esmentats, està documentada en la referència **[20]**).

Per a solucionar provisionalment aquest problema, el que s'ha fet en aquest treball és forçar la generació de l'arbre cada 6 minuts per tal de tenir una representació actual de l'escenari (i evitar que un mateix node mòbil aparegui com a veí de diferents nodes, per culpa de la informació obsoleta que mantenen els nodes). No obstant això, aquesta solució només és provisional ja que, quan la implementació del RPL del Contiki permeti actualitzar correctament les taules d'encaminament quan el DODAG es modifica per culpa dels desplaçaments dels nodes, aquesta reinicialització de l'arbre ja no serà necessària.

CAPÍTOL 7. CONCLUSIONS I TREBALL FUTUR

Aquest treball final de carrera planteja el disseny d'un sistema de representació gràfica de la posició aproximada del bestiar que pastura a la muntanya durant temporades llargues. Així, gràcies a la tecnologia actual, es pot facilitar la tasca dels ramaders i estalviar-los llargs trajectes recorrent la muntanya per trobar els animals.

El sistema que es proposa es basa en l'ús de petits dispositius de baix consum col·locats en posicions estratègiques de la zona de pastura i en cada animal. Així, cada animal té un identificador únic que el distingeix de la resta i que periòdicament envia la relació de veïns amb els que té visibilitat directa en aquell moment. Aquesta informació es transmet (a través de la xarxa multi-salt que formen els dispositius dels animals i els punts fixos) fins arribar a un node recol·lector de les dades. Aquest node, s'encarrega de processar tota la informació rebuda i representar-la gràficament. Així, s'obté una idea visual de la relació de veïnatge entre els animals i de la seva posició relativa en relació als nodes fixos que estan ubicats en localitzacions conegudes de la zona de pastura.

Per provar i validar la viabilitat de la proposta, s'ha utilitzat el simulador Cooja. Aquest simulador permet definir escenaris on els dispositius són sensors (*motes*) equipades amb el OS Contiki. Per a fer arribar la informació de cada animal i els veïns al node recol·lector, s'ha proposat utilitzar el protocol RPL (ja implementat al Contiki).

Aquest treball s'ha centrat en la definició, el disseny i la validació via simulació del sistema. No obstant, abans de poder fer el desplegament del sistema en un entorn real, encara queden alguns punts a resoldre:

a) Definició dels identificadors dels animals. Cal especificar quin format s'utilitzarà per a la identificació del bestiar, sobretot tenint en compte que hi pot haver ramats de pagesos diferents compartint la mateixa zona de pastura. Aquesta informació pot ser emmagatzemada en una base de dades de manera que l'aplicació que representi el mapa final sàpiga qui és el propietari de cadascun dels animals.

b) Simulació d'escenaris més complexos, on convisquessin diversos arbres RPL. El fet de mantenir múltiples arbres permetria escenaris amb un major nombre de nodes mòbils (animals). Els nodes arrel s'haurien de comunicar entre sí i fer arribar les dades al node recol·lector per a la seva representació.

c) Millora de la representació del sistema. Tenint en compte que la posició dels nodes fixos és coneguda, es podria afinar més a l'hora de representar la posició del nodes.

d) Programació de l'aplicació final que farà servir el pastor o el veterinari o l'encarregat de mantenir el control del bestiar per visualitzar la informació.

Pel que fa a l'ambientalització del sistema proposat, cal tenir en compte que s'utilitzarien dispositius de baixa potència i de baix consum col·locats a l'esquellot dels animals o en algun punt on no els fessin cap mal. Queda obert, no obstant, la incògnita de l'impacte de les ones electromagnètiques sobre la salut dels animals i les persones. Pel que fa als nodes fixos, també es proposa l'ús de dispositius de petites dimensions que, per tant, poden passar perfectament desapercebuts sense crear cap impacte visual sobre la zona on es desplegui el sistema

BIBLIOGRAFIA

[1] Routing for Low-Power and Lossy-Networks, RFC 6550:
1) Korte, K.D. *Evaluation of ContikiRPL Local Repairs*, "IPv6 over Low-power and Lossy Networks",15-24 (2011)
2) https://tools.ietf.org/html/rfc6550

[2] Xarxes WSN i aplicacions:
1) http://en.wikipedia.org/wiki/Wireless_sensor_network
2) https://www.pats.ua.ac.be/content/publications/2010/HeterWMN2010_Smolderen_DeCleyn.pdf

[3] Càlcul del *rank:*
Mula, I., Ferrari, G. i Firme, G. Estudio, *Análisis y Diseño de Redes de Sensore Inalámbricas con Contiki OS,* "Optimización de la memoria RAM y ROM", 39-40.

[4] Contiki Cooja
http://www.contiki-os.org/start.html

[5] C++ Api
http://www.cplusplus.com/reference/

[6] VMWare:
http://www.vmware.com/products/player/

[7] Simulador Cooja:
https://www.sics.se/search/content/contiki%20wiki%20index%20php%20Get%20started%20with%20Cooja%20simulator

[8] MicaZ mote datasheet:
http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

[9] Wismote datasheet:
http://www.aragosystems.com/images/stories/WiSMote/Doc/wismote_en.pdf

[10] Z1 mote datasheet:
http://www.zolertia.com/ti

[11] Tmote Sky datasheet:
http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf

[12] Datacheet MSP430
http://www.datasheetcatalog.com/datasheets_pdf/M/S/P/4/MSP430.shtml

[13] Descarregar el contiki:
http://sourceforge.net/projects/contiki/files/Instant%20Contiki/

[14] Alain, C. Pascal, B. Frédéric G. Serge, C. Marco, C. Steffen, R. Enrique, A. i Michel, S. *Mobile Ad Hoc Networks: Modeling, Simulation and Broadcast-based Applications,* "Mobility models in use", 85-88 (2007)

[15] C# Api
http://msdn.microsoft.com/en-us/library/d11h6832%28v=vs.71%29.aspx

[16] Documentació del Graphviz:
http://www.graphviz.org/doc/info/attrs.html

[17] Mula, I., Ferrari, G. i Firme, G. Estudio, *Análisis y Diseño de Redes de Sensore Inalámbricas con Contiki OS,* "Optimización de la memoria RAM y ROM", 58-60.

[18] Cooja scripts
https://github.com/contiki-os/contiki/wiki/Using-Cooja-Test-Scripts-to-Automate-Simulations

[19] Pàgines d'interès per Shell scripts:
http://cfajohnson.com/shell/resources/

[20] Documentació sobre les taules de *router* en Contiki:
1) http://comments.gmane.org/gmane.os.contiki.devel/9075
2) https://github.com/contiki-os/contiki/issues/114
3) https://github.com/contiki-os/contiki/pull/46
4) http://permalink.gmane.org/gmane.os.contiki.cvs/9600

ACRÒNIMS

6LoWPAN: IPv6 over Low Power Wireless Personal Area Networks.

DAG: Directed Acyclic Graph.

DAO: Destination Advertisement Object.

DIO: DODAG Information Onjects.

DIS: DODAG Information Solicitation.

DODAG: Destination-Oriented Directed Acyclic Graph.

GPS: Global Positioning System.

ID: IDentity.

IETF: Internet Engineering Task Force.

LED: Light Emitting Diode.

OS: Operating System.

RAM: Random Access Memory.

ROLL: Routing over Low-power and Lossy networks.

ROM: Read Only Memory.

RPL: Routing Protocol for Low Power and Lossy Networks.

TCP/IP: Transmission Control Protocol / Internet Protocol.

UDP: User Datagram Protocol.

WSN: Wireless Sensor Networks.